Lucrarea de laborator nr. 6

Interfeţe

Interfeţele sunt asemănătoare claselor abstracte care au toţi membri abstracţi: în cadrul unei interfeţe sunt definite metode, proprietăţi şi evenimente pe care o clasă care implementează interfaţa respectivă trebuie să le implementeze. Interfaţa nu poate conţine cîmpuri de date; în schimb, poate conţine proprietăţi, deoarece acestea sunt implementate prin metode. Interfeţele pot conţine de asemenea şi evenimente; atunci cînd un eveniment este parte a unei interfeţe el este parte a comportamentului pe care îl implementează interfaţa. În plus, definiţia unei interfeţe nu poate conţine modificatori de acces (de exemplu, `public`:) deoarece scopul interfeţelor este acela de a defini o funcționalitate (proprietăţile metodele şi evenimentele asociate unei clase care implementează interfața), nu are sens noţiunea de public sau privat în acest context.

Programarea interfeţelor reprezintă o tehnică foarte puternică. Practic, interfeţele descriu un comportament pe care clasele care le implementează îl vor moşteni; ele nu reprezintă capacitate de stocare pentru date şi nici implementare. Implementarea este responsabilitatea claselor care implementează interfaţa respectivă.
Structura generală a unei interfeţe este următoarea:

```
interface INumeInterfata
{
  tip_retur NumeProprietate
   {
     get;
     set;
   }
   void NumeMetoda();  ……
}
 //clasa care implementeaza interfata
class ClasaInterfata : INumeInterfata
 {
   //implementare explicita a membrilor interfetei
   tip_retur NumeProprietate
    {
     get {..//implementare..}
     set{..//implementare..}
    }

    void NumeMetoda ()
     {
      // implementare metoda
       ...
     }
 }
  static void Main()
    {
     // declararea unei instante a interfetei
     INumeInterfata obj = new ClasaInterfata ();
      // apel  membru interfata
     obj.NumeMetoda ();
         ...
    }
}
```

Definirea unei interfeţe se realizează utilizînd cuvîntul cheie `interface`. În rest, structura interfeţei este asemănătoare claselor. Asemenea claselor, o interfaţă este membră a unui spaţiu de nume şi conţine semnăturile membrilor: metode, proprietăţi şi evenimente (delegaţi). Mai poate conţine şi elemente de indexare, dacă este cazul. O interfaţă poate să fie derivată dintr-una una sau mai multe interfeţe de bază.

O clasă care implementează o interfaţă trebuie să implementeze în mod explicit **toţi** membri acelei interfeţe. Accesul la membri astfel implementaţi nu se va realiza prin intermediul unei instanţe a clasei ci prin intermediul unei referinţe la interfaţa pe care clasa o implemenează. În acest caz, prin intermediul referinţei definite se pot accesa doar membri interfeţei; pentru a accesa eventualii membri proprii ai clasei care implemenează interfaţa trebuie realizată o conversie explicită a referinţei către o referinţa la clasa respectivă.

Exerciţiu:

Pentru a modela situaţia reală în care comerciaţii fac oferte la produse şi/sau servicii „ambalate" într-un pachet vom extinde implementarea aplicaţiei dezvoltate în laboratoarele precedente prin construirea unei clase `Pachet`, clasă care este derivată din `ProdusAbstract`. Clasa abstractizează conceptul de pachet, care poate conţine unul sau mai multe produse şi/sau servicii. Astfel, vom crea o interfaţă `IPackageble` care să marcheze faptul că un `ProdusAbstract` poate sau nu să fie parte a unui pachet.

Creaţi interfaţa IPackageble

1. În fereastra **Solution Explorer** apăsaţi click dreapta pe soluţia (în care doriţi să creeaţi clasa), în cazul nostru în soluţia „entitati" , apoi **Add // Class** pentru a deschide fereastra **Add New Item.**

2. Selectaţi macheta **Interface** , setaţi numele interfeţei `IPackageble`, după care apăsaţi OK.

3. Adăugaţi metoda :

```
bool canAddToPackage(Pachet pachet);
```

Această metodă ne spune dacă un `ProdusAbstract` poate fi vândut (inclus) într-un pachet sau nu (returnând *true* respectiv *false*). Implementarea acestei metode se va face în clasele care implementează interfaţa `IPackageble`.

Creaţi clasa `Pachet`:

4. Creaţi clasa `Pachet` clasă care este derivată din `ProdusAbstract;` clasa abstractiza un pachet format din produse şi servicii (elemente `Produs` şi `Serviciu`)

```
public class Pachet : ProdusAbstract
```

5. Adăugaţi în cadrul clasei `Pachet` un tablou de elemente numit `elem_pachet` de tipul `IPackageble`. Tabloul de elemente reprezintă structura în care se vor memora elementele constituente ale pachetului (obiecte de tip `Produs` şi/sau `Serviciu`); se recomandă a fi utilizată pentru implementarea tabloului o clasă predefinită .NET (de exemplu, `List<T>`). La implementarea efectivă se va alege de modalitatea în care se vor putea constitui pachetele (se impun sau nu constrângeri în modalitatea de alcătuire: de exemplu : nu mai mult de 3 produse/servicii în total sau combinaţii de 2 produse cu 1 serviciu, etc.).

Implementaţi interfaţa IPackageble în clasele Produs şi Serviciu:

6. Modificaţi declaraţile claselor `Produs` şi `Serviciu` astfel încât acestea să implementeze interfaţa:

```
public class Produs : ProdusAbstract, IPackageble
public class Serviciu :ProdusAbstract, IPackageble
```

7. Pentru a corecta erorile datorate compilării trebuie să implementăm toate metodele interfeţei `IPackageble`, în cazul nostru trebuie să implementăm metoda `canAddToPackage` în clasa `Produs` şi `Serviciu`. Există mai multe alternative: scrierea directă a metodei în cadrul clasei (`Produs`) sau alegerea opţiunii *Implement Interface* (`Serviciu`)

8. De exemplu, scrierea directă implică implementarea următoarei funcţii în clasa `Produs`:

```
public bool canAddToPackage(Pachet pachet)
      {  return true;   }
```

9. Există şi alternativa de a genera automat codul implementării: apăsaţi click dreapta pe `IPackageble` în clasa `Serviciu` şi apoi **Quick Actions and Refactoring / Implement Interface,** lucru care va genera:

```
public bool canAddToPackage(Pachet pachet)
 {
     throw new NotImplementedException();
 }
```

Modificaţi apoi implementarea metodei astfel încît să returneze **true** (adică orice serviciu poate face parte din orice pachet – acest lucru se poate însă modifica ulterior dacă se doreşte impunerea anumitor restricţii).

11. Vizualizaţi rezultatul folosind ***Class diagram*** (Figura 6.1). In cadrul diagramei, câmpul `elem_pachet` al clasei `Pachet` este un tablou de elemente `IPackageble` şi a fost reprezentat pe diagramă utilizând opţiunea „***Show as (Collection) association***" (click dreapta pe câmp în diagrama de clase); se poate opta şi pentru varianta *„**Show as field**"*.

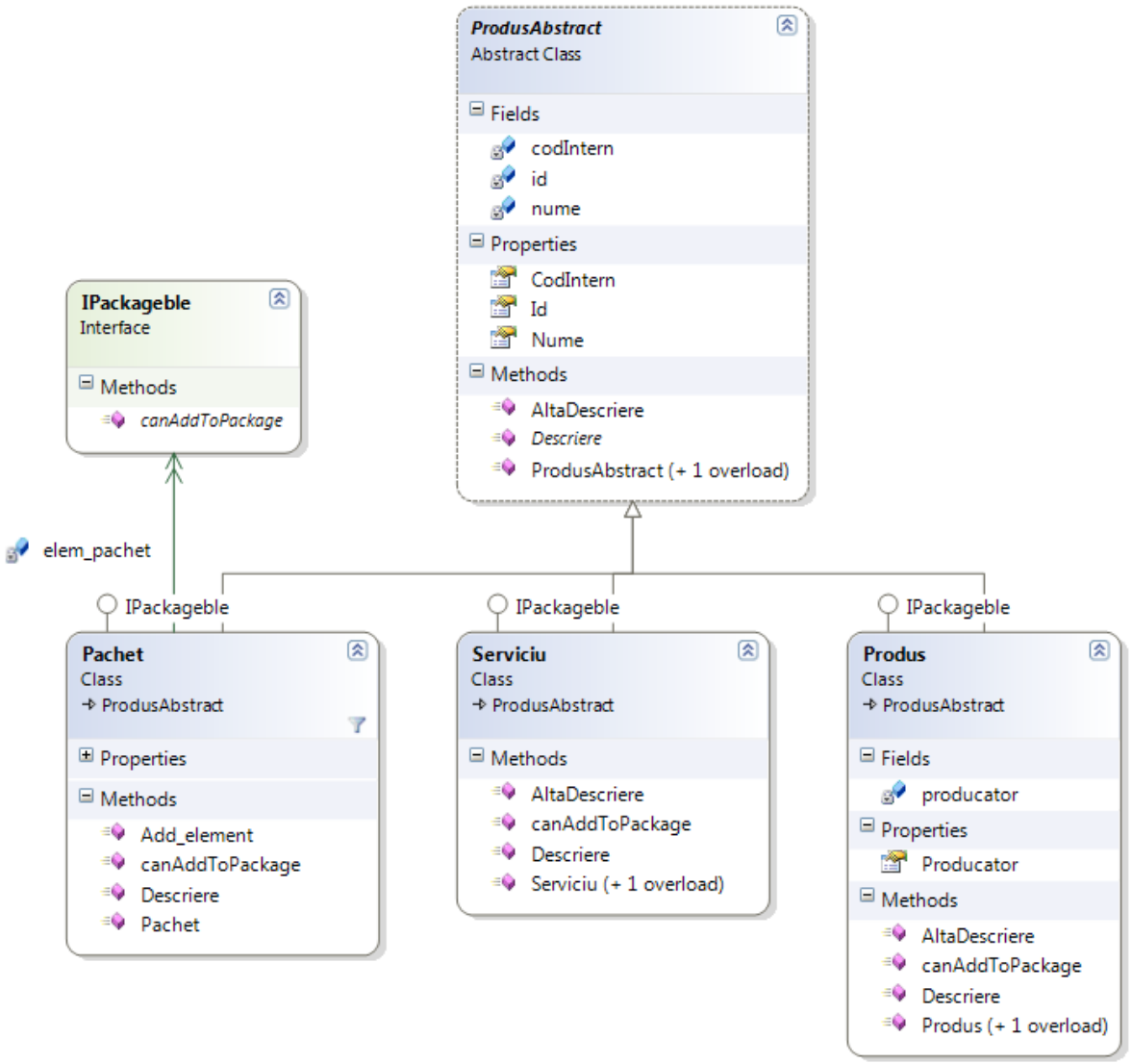
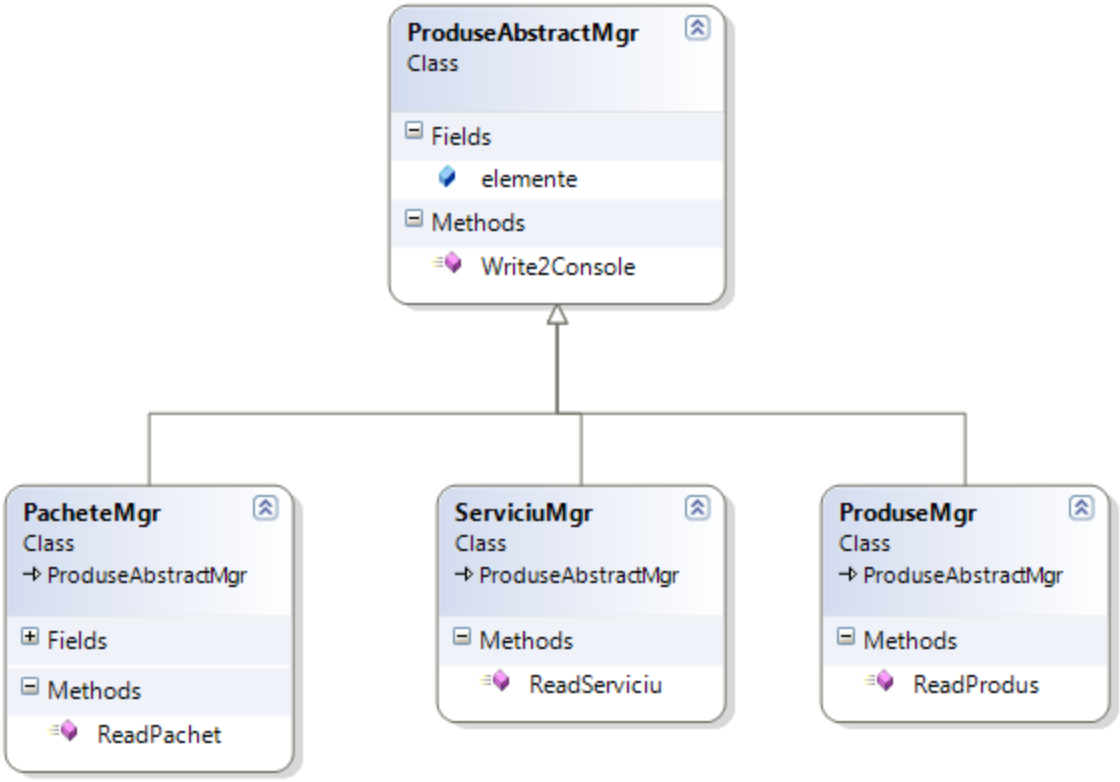


Figura 6.1: Implementarea ierarhiei de clase cu interfața `IPackageble`

Temă:

1. Pornind de la structura aplicaţiei prezentată în Figura 6.1, extindeţi aplicaţia prin contruirea unei clase manager pentru pachete cu numele `PachetMgr`, derivată din `ProdusAbstractMgr`. În mod similar cu managerii pentru produse/servicii, acesta va fi responsabil de gestionarea pachetelor. Includeți în cadrul acestuia opțiunea de citire a unui pachet (metoda `ReadPachet` – vezi diagrama de clase din Figura 6.2) şi utilizați-o apoi în `Main` pentru a citi pachete. Metoda `ReadPachet` va utiliza metodele corespunzătoare pentru citirea produselor şi/sau serviciilor din cadrul pachetului implementate în clasele `ProdusMgr` şi `ServiciuMgr`.



Adăugaţi în clasa `Program` cod pentru a permite utilizatorului să gestioneze pachete. Se citesc, prin intermediul managerilor de produse şi servicii, produsele şi serviciile aferente fiecărui pachet care se crează şi apoi se afişează pachetele create. Pentru implementarea colecţiilor de elemente/pachete puteţi utiliza colecţii din biblioteca .NET Framework.

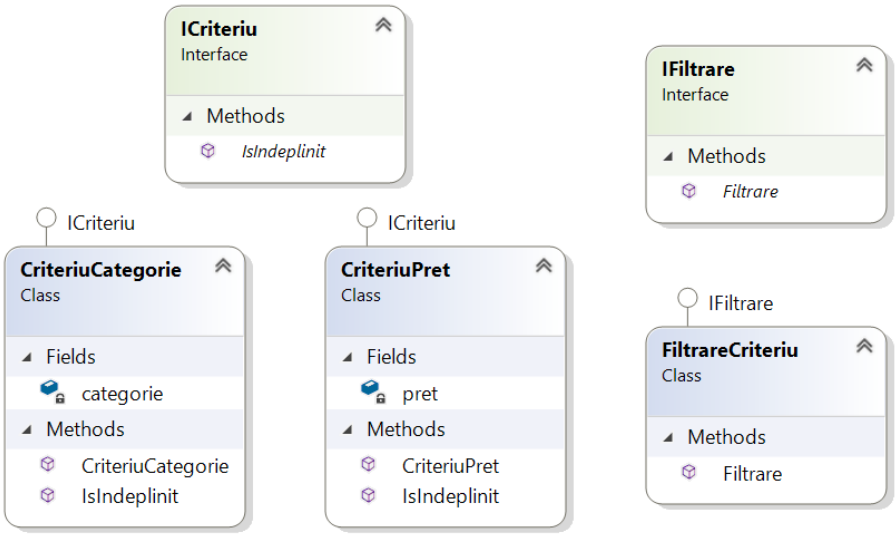
```
// crează un manager de pachet
PacheteMgr mgrPachet = new PacheteMgr();

Console.Write("Numar de pachete dorit:");
int nrPack = int.Parse(Console.ReadLine());

//citeste pachetele utilizând managerii de produse şi servicii
for (int i = 0; i < nrPack; i++) mgrPachet.ReadPachet();

//afiseaza pachetele
mgrPachet.Write2Console();
```

2. Modificaţi implementările funcţiei `canAddToPackage` astfel încât într-un pachet să putem avea maxim un produs şi un număr nelimitat de servicii (sau, alte variante de configuraţii – de exemplu pentru fiecare pachet, acesta poate fi format dintr-un numar `p` de produse şi un numar `s` de servicii). In ce masură soluţia permite adăugarea de alte constrângeri în crearea pachetelor?

3. Adăugaţi câmpul `Pret` pentru fiecare `Produs` şi `Serviciu` din cadrul unui pachet. Calculaţi preţul total al fiecărui pachet şi afişaţi pachetele în ordinea crescătoare a preţului total. Utilizaţi metoda `Sort` a colecţiei `List<T>` pentru sortarea pachetelor.

4. Completaţi (extindeţi) aplicaţia dezvoltată pentru gestionarea produselor, serviciilor şi pachetelor dezvoltată până în acest moment cu posibilitatea de a realiza diferite tipuri de filtrări asupra tabloului de elemente construit în cadrul aplicaţiei:
   - filtrarea după categorie => toate elementele care aparţin unei categorii
   - filtrarea după preţ => toate elementele care au preţul egal/mai mic/mai mare decât un preţ dat
   - …etc.

Implementaţi în acest sens o variantă care să permită extinderea cu uşurinţă a criteriilor de filtrare (adăugarea unui nou criteriu) precum şi a modalităţii de implementare a filtrelor (variantă extensibilă). O posibilă arhitectură în acest sens poate fi următoarea:



In cadrul acesteia, se utilizează două interfeţe:

- `ICriteriu` va fi implementată de toate clasele care definesc un criteriu specific de filtrare. Fiecare clasă va implementa în mod specific metoda *IsIndeplinit*
- `IFiltrare` va fi implementată de clasele care definesc o anumita modalitate de filtrare; un exemplu de implementare ar putea fi următorul:

```
public IEnumerable<ProdusAbstract> Filtrare(colectia_de_elemente, criteriul_de_filtrare)
        {
          //returneaza acele elemente din colectia_de_elemente care satisfac  criteriul_de_filtrare
        }
```

Adaptaţi exemplificarea de mai sus în contextul aplicaţiei de la laborator. Analizaţi rolul pe care implementarea bazată pe interfeţe aduce un plus de generalitate problemei faţă de alte potenţiale versiuni de implementare particulare. Analizaţi în ce masură s-ar putea introduce în cadrul arhitecturii de clase de mai sus o filtrare după mai multe criterii (categorie şi preţ, sau alte variante) şi realizaţi o variantă de implementare.